flex布局

目录
一、 Flex布局是什么？
二、基本概念
三、容器的属性
3.1 flex-direction属性
3.2 flex-wrap属性
3.3 flex-flow
3.4 justify-content属性
3.5 align-items属性
3.6 align-content属性
四、项目的属性
4.1 order属性
4.2 flex-grow 属性
4.3 flex-shrink 属性
4.4 flex-basis 属性
4.5 flex 属性
4.6 align-self 属性
Flex布局实例教程
一、骰子的布局
1.1 单项目：首行左对齐
1.2 单项目：首行居中对齐
1.3 单项目：首行右对齐
1.4 单项目：中行左对齐
1.5 单项目：垂直居中对齐
1.6 单项目：尾行居中对齐
1.7 单项目：尾行右对齐
1.8 双项目：首行两端对齐
1.9 双项目：首列两端对齐
1.10 双项目：中列两端对齐

1.11 双项目：尾列两端对齐
1.12 双项目：首行左对齐，中行居中对齐
1.13 双项目：首行左对齐，尾行右对齐
1.14 三项目：对角线布局
1.15 四项目：首行排满，尾行右对齐
1.16 四项目：四角对齐
1.17 六项目：首行排满，尾行排满
1.18 六项目：首列排满，尾列排满
1.19 六项目：首行两端对齐，中行居中，尾行排满
1.20 九项目
二、网格布局
2.1 基本网格布局
2.2 百分比布局
三、圣杯布局
四、输入框的布局
五、悬挂式布局
六、固定的底栏
两栏式布局
参考链接

网页布局（ layout）是 CSS的一个重点应用。



传统的布局有下面几种。

display: block;
display: inline;
display: list-item;
display: table;

display属性搭配 position属性和 float属性，是最常见的布局解决方案。

但是，这个方案基于 盒状模型 ，对于那些特殊位置的布局非常不方便。比如， 垂直居中就很不容易实现，代码好像一种破解，而不是正常的解决。

```
.element {
  position: relative;
  top: 50%;
  transform: translateY(-50%);
}
```

2009年，W3C提出了一种新的方案——Flex布局，用来简便地、完整地、响应式地实现各种页面布局。经过5年多的发展，Flex布局几乎得到了所有浏览器的支持。这意味着，你现在就能很安全地使用这项功能。

可以毫无疑问地说，Flex布局是未来的首选方案。本文介绍它的语法，主要是翻译和参考了下面两篇文章：A Complete Guide to Flexbox和 A Visual Guide to CSS3 Flexbox Properties。

一、Flex布局是什么?

Flex 是 flexible box 的缩写，意为弹性的盒状布局，中文可以译为“弹性布局”。它的设计目的是为盒状模型提供最大的灵活性，可以在不知道容器大小或者容器大小可变的情况下，指定布局方式。

任何一个容器都可以指定为 Flex 布局。

```
.box {
  display: flex;
}
```

行内元素也可以使用 Flex 布局。

```
.box {
  display: inline-flex;
}
```

Webkit内核的浏览器，必须加上-webkit前缀。



```
.box{
  display: -webkit-flex; /* Safari */
  display: flex;
}
```

注意，设为 Flex布局以后，子元素的 float、clear和 vertical-align属性将失效。

二、基本概念

采用 Flex 布局的元素，称为 Flex 容器（ flex container），简称 “容器 ”。它的所有子元素自动成为容器成员，称为 Flex 项目（ flex item），简称 “项目 ”。

容器默认存在两根轴：水平的主轴（ main axis）和垂直的交叉轴（ cross axis）。主轴的开始位置（与边框的交叉点）叫做 main start，结束位置叫做 main end；交叉轴的开始位置叫做 cross start，结束位置叫做 cross end。

项目默认沿主轴排列，在主轴占据的空间叫做 main size，在交叉轴占据的空间叫做 cross size。

三、容器的属性

以下 6 个属性设置在容器上。

> - flex-direction
> - flex-wrap
> - flex-flow
> - justify-content
> - align-items
> - align-content

3.1 flex-direction属性

`flex-direction` 属性决定主轴的方向（即项目的排列方向）。











```
.box {
  flex-direction: row | row-reverse | column | column-reverse;
}
```

它可能有 4个值。

- `row`（默认值）：主轴为水平方向，起点在左端。
- `row-reverse`：主轴为水平方向，起点在右端。
- `column`：主轴为垂直方向，起点在上沿。
- `column-reverse`：主轴为垂直方向，起点在下沿。

3.2 flex-wrap属性

默认情况下，项目都排在一条线（又称 “轴线 ”）上。 `flex-wrap`属性定义，如果一条轴线排不下，如何换行。

```
.box{
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

它可能取三个值。

（1） `nowrap`（默认）：不换行。

（2） `wrap`：换行，第一行在上方。



（3） `wrap-reverse`：换行，第一行在下方。

3.3 flex-flow

`flex-flow`属性是 `flex-direction`属性和 `flex-wrap`属性的简写形式，默认值为 `row nowrap`。

```
.box {
  flex-flow: <flex-direction> || <flex-wrap>;
}
```

下面是一个例子。

```
div {
  display: flex;
  flex-flow: row wrap;
}
```

3.4 justify-content属性

`justify-content`属性定义了项目在主轴上的对齐方式。

```
.box {
  justify-content: flex-start | flex-end | center | space-between | space-around;
}
```

它可能取 5个值，具体对齐方式与轴的方向有关。下面假设主轴为从左到右。

- `flex-start`（默认值）：左对齐
- `flex-end`：右对齐
- `center`： 居中
- `space-between`：两端对齐，项目之间的间隔都相等，项目与边框没有间隔。
- `space-around`：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

3.5 align-items属性

`align-items`属性定义项目在交叉轴上如何对齐。

```
.box {
  align-items: flex-start | flex-end | center | baseline | stretch;
}
```

它可能取 5个值。具体的对齐方式与交叉轴的方向有关，下面假设交叉轴从上到下。

- `flex-start`：交叉轴的起点对齐。
- `flex-end`：交叉轴的终点对齐。
- `center`：交叉轴的中点对齐。
- `baseline`：项目的第一行文字的基线对齐。
- `stretch`（默认值）：如果项目未设置高度或设为 auto，将占满整个容器的高度。

3.6 align-content属性

`align-content`属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

```
.box {
  align-content: flex-start | flex-end | center | space-between | space-around | stretch;
}
```

该属性可能取 6个值。

> - `flex-start`：与交叉轴的起点对齐。
> - `flex-end`：与交叉轴的终点对齐。
> - `center`：与交叉轴的中点对齐。
> - `space-between`：与交叉轴两端对齐，轴线之间的间隔平均分布。
> - `space-around`：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。
> - `stretch`（默认值）：轴线占满整个交叉轴。





四、项目的属性

以下 6个属性设置在项目上。

- `order`
- `flex-grow`
- `flex-shrink`
- `flex-basis`
- `flex`
- `align-self`

4.1 order属性

项目在主轴的排列顺序，默认是其出现的顺序。 `order`

属性可以改变这一点，让后面出现的项目，排到前面去。

`order` 属性定义项目的排列顺序。数值越小，排列越靠前，默认为 `0` 。

```
.item {
  order: <integer>;
}
```

4.2 flex-grow 属性

`flex-grow` 属性定义项目的放大比例，默认为 0，即如果主轴存在剩余空间，也不放大。

```
.item {
  flex-grow: <number>; /* default 0 */
}
```

如果所有项目的 `flex-grow` 属性都为 1，则它们将等分剩余空间（如果有的话）。如果一个项目的 `flex-grow` 属性为 2，其他项目都为 1，则前者占据的剩余空间将比其他项目多一倍。因此， `flex-grow` 也可以理解成放大的速率， `2` 的增长速率比 `1` 快一倍。

`flex-grow` 的值是一种比例关系，所有项目都设成 `1`，与都设成 `2` 或 `0.5`，结果是一样的。

4.3 flex-shrink 属性

`flex-shrink` 属性定义了项目的缩小比例，默认为 1，即如果空间不足，该项目将缩小。

```
.item {
  flex-shrink: <number>; /* default 1 */
}
```

如果所有项目的 `flex-shrink` 属性都为 1，当空间不足时，都将等比例缩小，否则项目宽度大于容器宽度，会发生溢出。如果一个项目的 `flex-shrink` 属性为 0，其他项目都为 1，则空间不足时，前者不缩小。

负值对该属性无效。

4.4 flex-basis 属性

`flex-basis` 属性定义了在分配多余空间之前，项目占据的主轴空间（main size），即项目本身的尺寸。浏览器根据这个属性，计算主轴是否有剩余空间。它的默认值为 `auto`，即项目的本来大小。

```
.item {
  flex-basis: <length> | auto; /* default auto */
}
```

它可以设为跟 `width` 或 `height` 属性一样的值（比如 350px，20%，5em等等），则项目将占据固定空间。

4.5 flex 属性

`flex` 属性是 `flex-grow`，`flex-shrink` 和 `flex-basis` 的简写，默认值为 `0 1 auto`。后两个属性可选。

```
.item {
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
}
```



该属性有两个快捷值: `auto` (代表 `1 1 auto`) 和 none (代表 `0 0 auto`)。

4.6 align-self 属性

`align-self`属性允许单个项目有与其他项目不一样的对齐方式，可覆盖 `align-items`属性。默认值为 `auto`，表示继承父元素的 `align-items`属性，如果没有父元素，则等同于 `stretch`。

```
.item {
  align-self: auto | flex-start | flex-end | center | baseline | stretch;
}
```

该属性可能取 6个值，除了 `auto`，其他都与 `align-items`属性完全一致。

（完）

Flex布局实例教程

上一篇文章，介绍了 Flex布局的语法。

今天介绍，常见布局的 Flex写法。你会看到，不管是什么布局， Flex往往都可以几行命令搞定。

下面的内容，我只列出代码，详细的语法解释请查阅 《Flex布局语法教程》。

我主要参考了 Landon Schropp的文章和 Solved by Flexbox。

一、骰子的布局

如果不加说明，本节的 HTML模板一律如下。

```
<div class="box">
  <span class="item"></span>
</div>
```







上面代码中，div元素是Flex容器，span元素是Flex项目。如果有多个项目，就要添加多个span元素，以此类推。

1.1 单项目：首行左对齐

Flex布局默认就是首行左对齐，所以一行代码就够了。

```
.box {
  display: flex;
}
```

1.2 单项目：首行居中对齐

```
.box {
  display: flex;
  justify-content: center;
}
```

1.3 单项目：首行右对齐

```
.box {
  display: flex;
  justify-content: flex-end;
}
```

1.4 单项目：中行左对齐



```
.box {
  display: flex;
  align-items: center;
}
```

1.5 □□□□□□□□



```
.box {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

1.6 □□□□□□□□



```
.box {
  display: flex;
  justify-content: center;
  align-items: flex-end;
}
```

1.7 □□□□□□□



```
.box {
  display: flex;
  justify-content: flex-end;
  align-items: flex-end;
}
```

1.8 □□□□□□□□



```
.box {
  display: flex;
  justify-content: space-between;
}
```

1.9 □□□□□□□□



```
.box {
  display: flex;
  flex-direction: column;
  justify-content: space-between;
}
```

1.10 □□□□□□□□



```
.box {
  display: flex;
  flex-direction: column;
```

```
  justify-content: space-between;
  align-items: center;
}
```

1.11 □□□□□□□□□



```
.box {
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: flex-end;
}
```

1.12 □□□□□□□□□□□□□□



```
.box {
  display: flex;
}

.item:nth-child(2) {
  align-self: center;
}
```

1.13 □□□□□□□□□□□□□□



```
.box {
  display: flex;
  justify-content: space-between;
}

.item:nth-child(2) {
  align-self: flex-end;
}
```

1.14 □□□□□□□□



```
.box {
  display: flex;
}

.item:nth-child(2) {
  align-self: center;
}

.item:nth-child(3) {
  align-self: flex-end;
}
```

1.15 □□□□□□□□□□□□



```
.box {
  display: flex;
  flex-wrap: wrap;
  justify-content: flex-end;
  align-content: space-between;
```

```
}
```

1.16 □□□□□□□



HTML□□□□□

```
<div class="box">
  <div class="column">
    <span class="item"></span>
    <span class="item"></span>
  </div>
  <div class="column">
    <span class="item"></span>
    <span class="item"></span>
  </div>
</div>
```

CSS□□□□□

```
.box {
  display: flex;
  flex-wrap: wrap;
  align-content: space-between;
}

.column {
  flex-basis: 100%;
  display: flex;
  justify-content: space-between;
}
```

1.17 □□□□□□□□□□□□



```
.box {
  display: flex;
  flex-wrap: wrap;
  align-content: space-between;
}
```

1.18 □□□□□□□□□□□



```
.box {
  display: flex;
  flex-direction: column;
  flex-wrap: wrap;
  align-content: space-between;
}
```

1.19

□□□□□□□□□□□□□□□□□□□□



HTML□□□□□

```
<div class="box">
  <div class="row">
    <span class="item"></span>
    <span class="item"></span>
    <span class="item"></span>
  </div>
  <div class="row">
    <span class="item"></span>
  </div>
  <div class="row">
```

```
      <span class="item"></span>
      <span class="item"></span>
    </div>
  </div>
```

CSS□□□□□

```
.box {
  display: flex;
  flex-wrap: wrap;
}

.row{
  flex-basis: 100%;
  display:flex;
}

.row:nth-child(2){
  justify-content: center;
}

.row:nth-child(3){
  justify-content: space-between;
}
```

1.20 □□□



```
.box {
  display: flex;
  flex-wrap: wrap;
}
```

□□□□□□



2.1 基本网格布局

容器平均分配空间。

HTML代码如下。

```
<div class="Grid">
  <div class="Grid-cell">…</div>
  <div class="Grid-cell">…</div>
  <div class="Grid-cell">…</div>
</div>
```

CSS代码如下。

```
.Grid {
  display: flex;
}

.Grid-cell {
  flex: 1;
}
```

上面这样的设置，不管父容器之中有多少个项目，都会平分宽度。

遇到小屏幕时，可以使用 media query，让项目占满一行。

```
@media (max-width: 400px) {
  .flex-grid {
    display: block;
  }
}
```

网格之间的间隔，可以像下面这样设置。

```
.flex-grid-thirds {
  display: flex;
  justify-content: space-between;
```

```
}
.flex-grid-thirds .col {
  width: 32%;
}
```

2.2 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□



HTML□□□□□

```
<div class="Grid">
  <div class="Grid-cell u-1of4">…</div>
  <div class="Grid-cell">…</div>
  <div class="Grid-cell u-1of3">…</div>
</div>
```

```
.Grid {
  display: flex;
}

.Grid-cell {
  flex: 1;
}

.Grid-cell.u-full {
  flex: 0 0 100%;
}

.Grid-cell.u-1of2 {
  flex: 0 0 50%;
}

.Grid-cell.u-1of3 {
  flex: 0 0 33.3333%;
```

```
}

.Grid-cell.u-1of4 {
  flex: 0 0 25%;
}
```

三、圣杯布局

圣杯布局（Holy Grail Layout）指的是一种最常见的网站布局。页面从上到下，分成三个部分：头部（header），躯干（body），尾部（footer）。其中躯干又水平分成三栏，从左到右为：导航、主栏、副栏。

HTML代码如下。

```
<body class="HolyGrail">
  <header>…</header>
  <div class="HolyGrail-body">
    <main class="HolyGrail-content">…</main>
    <nav class="HolyGrail-nav">…</nav>
    <aside class="HolyGrail-ads">…</aside>
  </div>
  <footer>…</footer>
</body>
```

CSS代码如下。

```
.HolyGrail {
  display: flex;
  min-height: 100vh;
  flex-direction: column;
}

header,
footer {
  flex: 1;
}
```

```
.HolyGrail-body {
  display: flex;
  flex: 1;
}

.HolyGrail-content {
  flex: 1;
}

.HolyGrail-nav, .HolyGrail-ads {
  /* □□□□□□□□□12em */
  flex: 0 0 12em;
}

.HolyGrail-nav {
  /* □□□□□□□ */
  order: -1;
}
```

□□□□□□□□□□□□□□□□□□□

```
@media (max-width: 768px) {
  .HolyGrail-body {
    flex-direction: column;
    flex: 1;
  }
  .HolyGrail-nav,
  .HolyGrail-ads,
  .HolyGrail-content {
    flex: auto;
  }
}
```

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□





HTML代码如下。

```
<div class="InputAddOn">
  <span class="InputAddOn-item">…</span>
  <input class="InputAddOn-field">
  <button class="InputAddOn-item">…</button>
</div>
```

CSS代码如下。

```
.InputAddOn {
  display: flex;
}

.InputAddOn-field {
  flex: 1;
}
```

五、悬挂式布局

有时，主栏的左侧或右侧，需要添加一个图片栏。

HTML代码如下。

```
<div class="Media">
  <img class="Media-figure" src="" alt="">
  <p class="Media-body">…</p>
</div>
```

CSS代码如下。

```
.Media {
  display: flex;
  align-items: flex-start;
}

.Media-figure {
  margin-right: 1em;
}

.Media-body {
  flex: 1;
}
```

六、固定的底栏

有时，页面内容太少，无法占满一屏的高度，底栏就会抬高到页面的中间。

这时可以采用 Flex布局，让底栏总是出现在页面的底部。

HTML代码如下。

```
<body class="Site">
  <header>…</header>
  <main class="Site-content">…</main>
  <footer>…</footer>
</body>
```

CSS代码如下。

```
.Site {
  display: flex;
  min-height: 100vh;
  flex-direction: column;
}

.Site-content {
```

```
  flex: 1;
}
```

□□□□

□□□□□□□□□□ Flexbox □□□□□□

```
/*
  HTML □□□□
  <div class="wrapper">
    <section class="main-column" />
    <aside class="sidebar" />
  </div>
*/

.wrapper {
  display: flex;
}

.main-column {
  width: 70%;
  padding-right: 2rem;
}

.sidebar {
  width: 30%;
}
```

□□□

□□□□

- Getting Dicey With Flexbox, by Landon Schropp
- A Complete Guide to Flexbox
- Dimitar Stojanov, A Visual Guide to CSS3 Flexbox Properties

---

□□□ #2
□ □□□ □□□ 2017-02-16 16:44:53 UTC

□ □□□ □□□ 2022-06-22 16:35:00 UTC